INTRODUCTION TO GARBAGE COLLECTION IN JAVA

- In older languages like C++, the programmer is responsible for creating and destroying objects. If they forget to destroy unused objects, memory leaks can occur, and the application may crash.
- In Java, the programmer only needs to create objects. The destruction of unused objects is handled automatically by the Garbage Collector (GC).
- The Garbage Collector runs in the background and removes objects that are no longer needed, reducing the chances of memory-related failures.
- Main goal of GC: Free up memory by destroying useless objects (objects that are no longer referenced).

When Is An Object Eligible For Garbage Collection:

An object becomes eligible for GC when **no live thread can access it** (i.e., no reference points to it).

Ways to Make an Object Eligible for GC

- **Assign null to the reference**

```java
String str = new String("Hello");
str = null; // "Hello" object is now eligible for GC
```

- **Reassign the reference to another object**

```java
String str = new String("Hello");
str = new String("World"); // "Hello" object is now eligible for GC
```

- **Objects created inside a method (local objects)**

  When the method ends, local variables go out of scope.

```java
void demo() {
    String temp = new String("Temporary");
    // After method ends, "Temporary" object is eligible for GC
}
```

- Isolated objects (no references from any reachable object)

```java
class Node {
    Node next;
}
Node n1 = new Node();
Node n2 = new Node();
n1.next = n2;
n2.next = n1;
n1 = null;
n2 = null; // Both nodes are now isolated and eligible for GC
```

REQUESTING JVM TO RUN GC:

BY SYSTEM CLASS:

System class contains a static method GC for this purpose.

Example: System.gc();

BY RUNTIME CLASS:

- A java application can communicate with jvm by using Runtime object.
- Runtime class is a singleton class present in java.lang. Package.
- We can create Runtime object by using factory method getRuntime().

Example:
Runtime r=Runtime.getRuntime();Once we got Runtime object we can call the following methods on that object.
freeMemory(): returns the free memory present in the heap.
totalMemory(): returns total memory of the heap. gc(): for requesting jvm to run gc.

☑ Key Points for Interviews

- GC does not guarantee immediate cleanup.
- Eligibility ≠ Immediate destruction.
- Two main ways to request GC: System.gc() and Runtime.getRuntime().gc().

GARBAGE COLLECTION IN JAVA

WHAT IS AN OBJECT ELIGBLE FOR GC?

- An object is eligible when no live references point to it.
- In Java, the programmer only needs to create objects.GC destroys unused objects.
- GC runs in the background to free memory

WAYS TO MAKE AN OBJECT ELIGIBLE FOR GC

Assign null to a reference

An object is eligible when no live references point to it.

Reasssign a reference


```
String s = new String('Hello');
s = null
```

Object created inside a method


```
void myMethod() {
String temp = newString(Hi');
}
```

REQUESTING JVM TO RUN GC

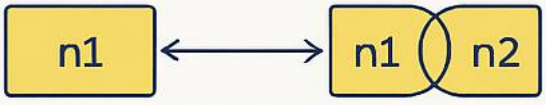
- Using System class

```
System.gc();
```

Object created inside a method


```
Node n1 = new Node()
Node n2 = new Node()
n1.next = n2
n2.next = n1
```

Island of isolation




```
Node n1 = new Node()
Node n2 = new Node()
n1.next = n2
n2.next = n1
```

GC does not guarantee immediate cleanup.

FINALIZATION:

- Before destroying an object, Garbage Collector (GC) calls the finalize() method to perform cleanup activities.
- If a class overrides finalize(), that method runs; otherwise, Object class's finalize() runs (which is empty).

```
protected void finalize() throws Throwable
```

Key Points

- GC calls finalize() before object destruction.
- We can call finalize() explicitly, but it acts like a normal method call (object is NOT destroyed).
- GC does NOT guarantee when finalize() will run (or even if it will run).

CASE 1: OBJECT ELIGIBLE FOR GC

- If String object becomes eligible for GC, **String class finalize()** runs (empty implementation).
- If Test object becomes eligible, **Test class finalize()** runs.

```
class Test {
    public static void main(String[] args) {
        String s = new String("Hello");
        s = null;
        System.gc();
        System.out.println("End of main");
    }
    public void finalize() {
        System.out.println("finalize() executed");
    }
}
```

Output: (Only String's empty finalize() executed)

```
End of main
```

Example 2: Test object

```
class Test {
    public static void main(String[] args) {
        Test t = new Test();
        t = null;
        System.gc();
        System.out.println("End of main");
    }
    public void finalize() {
        System.out.println("finalize() executed");
    }
}
```

Output:

```
finalize() executed
End of main
```

CASE 2: EXPLICIT CALL TO FINALIZE():

Acts like a normal method call; object is NOT destroyed.

```java
class Test {
    public static void main(String[] args) {
        Test t = new Test();
        t.finalize();
        t.finalize();
        t = null;
        System.gc();
        System.out.println("End of main");
    }
    public void finalize() {
        System.out.println("finalize() called");
    }
}
```

Output:

```
finalize() called
finalize() called
End of main
```

Interview Tips

- finalize() is **deprecated in Java 9** and removed in later versions. Use **try-with-resources** or **AutoCloseable** instead.
- GC is **not guaranteed** to run immediately.
- Never rely on finalize() for critical cleanup.

Note: In Servlets we can call destroy() method explicitly from init() and service() methods. Then it will be executed just like a normal method call and Servlet object won't be destroyed.

CASE 3:

- finalize() method can be call either by the programmer or by the GC .
- If the programmer calls explicitly finalize() method and while executing the finalize() method if an exception raised and uncaught then the program will be terminated abnormally.
- If GC calls finalize() method and while executing the finalize()method if an exception raised and uncaught then JVM simply ignores that exception and the program will be terminated normally.

CASE 4:

- On any object GC calls finalize() method only once.